

1 XML DATA ENCODING AND DECODING

2 FIELD OF THE INVENTION

3 The present invention relates to a technique for encoding
4 and decoding data. It more particularly relates to
5 improving the compression and decompression rates when XML
6 (Extensible Markup Language) data is encoded and decoded.

7 BACKGROUND

8 Recently, attention has been drawn to XML as a data
9 expression means for use on the Internet. XML is an
10 extensible meta language, and a user can uniquely define
11 its grammar. In addition, while XML can provide logical
12 meaning for each element, it is much easier to use than
13 HTML (Hypertext Markup Language) for data processing. It
14 is therefore anticipated that XML will become the standard
15 expression method and will be used for the structural
16 languages that will be employed for the exchange of
17 e-commerce documents, for example. Note that the
18 specification for XML is contained in "W3C.Extensible
19 Markup Language (XML) 1.0, 1998,
20 <http://www.w3.org/TR/REC-xml>".

21 Since characters are used to write XML data, its
22 readability is high, as is its redundancy. Specifically,
23 the meaning of an element is written mainly between start
24 and end tags, and can be easily understood merely by

1 referring to the contents, which, to reiterate, are
 2 written using characters. However, as characters are used
 3 to write all the contents, the total number of characters
 4 used is increased, and overall, the amount of data (the
 5 character count) required for an XML document is also
 6 increased. Thus, since a large number of characters are
 7 used, either a large memory capacity is required for the
 8 storage of data, and/or accompanying the increase in the
 9 amount of data, there is an increase in physical labor and
 10 time costs when the data is transferred via a network.
 11 Therefore, it would be convenient were XML data encoded
 12 (or compressed) to shorten the length of the code
 13 employed.

14 A variety of well known data compression methods are
 15 presently available, and include: run-length coding,
 16 Huffman coding, arithmetic coding and LZ77. Example
 17 in-detail descriptions of these compression methods may be
 18 found in: "A method for the construction of
 19 minimum-redundancy codes", Huffman, D.A., Proc. of the
 20 IRE, September, 1952; "The Data Compression Book", Mark
 21 Nelson and Jean Loup Gailly, Second Edition, M&T Books
 22 1996; and "A universal algorithm for sequential data
 23 compression", Jacob Ziv and Abraham Lempel, IEEE
 24 Transactions on Information Theory, May, 1997.

25 However, these compression methods were not specially
 26 prepared XML, and when used for XML data, compression
 27 efficiency is not always high. Example specialized
 28 compression methods for XML data are: XMill, described in

1 "XMill: an Efficient Compressor for XML data, 1999,
 2 <http://www.research.att.com/sw/tools/xmill/>", D. Suciu and
 3 H. Liefke; XMLZip, described in "XMLZip, 1999,
 4 <http://www.xmls.com/products/xmlzip/xmlsip.html>", XML
 5 Solutions Corp.; and XComp, described in "Study for an XML
 6 document compression algorithm using DTD", Kousaku Ikawa,
 7 graduation thesis prepared for the Information engineering
 8 course given by the Technology department of the Tokyo
 9 Institute of Technology, February, 2000.

10 According to the XMill reference, the content (text)
 11 portion of each element is extracted from XML data, and
 12 this extracted portion is referred to as a container.
 13 Then the structural portion is encoded using numerals, and
 14 subsequently, the text portion for each container is
 15 compressed using a compression method such as LZ77.
 16 Basically, data compression can be performed by an
 17 application without additional information, such as
 18 parameters, being required. As needed, a compression
 19 method for each container can be designated by setting a
 20 parameter, a process that increases compression
 21 efficiency. Further, since C is used to implement XMill,
 22 the compression speed is high.

23 According to the XMLZip reference, the depth of a root
 24 element is designated, and the designated portion is
 25 separated from a document element, following which ZIP is
 26 used to compress the remaining portion. Therefore, since
 27 the root element is not encoded operations on it can be
 28 performed directly. Further, since only the portion that

1 is not so used is compressed, rapid document access is
2 possible. It should be noted, however, that the
3 compression efficiency provided by XMLZip is lower than
4 that provided by XMill.

5 According to the XComp reference, of the structural
6 portions that constitute XML data, a portion that is
7 uniquely determined by employing DTD (Document Type
8 Definition) is not encoded, and only that portion which
9 can not be uniquely determined is compressed. Compression
10 of the text portion is performed in the same manner as it
11 is for XMill. That is, for data compression, the
12 following procedures are performed. (1) XML data is
13 divided into structure and content; (2) DTD is used to
14 generate a push-down automaton (PDA); (3) PDA is used to
15 generate an encoding transducer for encoding the
16 structural portion; (4) numbers that are allocated for
17 the individual nodes of the encoding transducer are output
18 by continuously transferring the automaton while the
19 structure is encoded; and (5) a method such as LZ77 is
20 used to compress the code obtained for the structure and
21 the contents of the elements, following which the
22 compressed XML document is output.

23 Of the specialized compression methods employed for XML
24 documents, a comparatively higher compression efficiency
25 can be obtained with XComp, which does not encode part of
26 the structural portion.

27 Even though the compression of XML documents, XComp is the

1 superior method. However, according to a study performed
2 by the present inventor, when XComp is used to process XML
3 data having a specific structure, compression efficiency
4 is reduced. That is, when the "?" operator or the "*" operator (includes the "+" operator) are employed for
5 elements, compression efficiency is reduced.

7 The "?" operator is the one that is attached to the child
8 element of a specific element in an element type
9 declaration when the child element does not appear or
10 appears only one time. According to XComp, when the "?" operator appears the "?" operator is represented by
11 multiple choices each of which is shifted from one
12 specific state to another, and an index is provided for
13 the choice. Thereafter, when XComp is executed, an index
14 provided for a selected choice is output, and since the
15 number of available choices varies in consonance with how
16 many "?" operators follow an initial "?" operator, when n
17 "?" operators persist, for example, n+1 choices are
18 present for the first "?" operator. Therefore, n+1
19 indexes are required, and $O(\log n)$ bits are needed to
20 represent one index. Thus, when all the elements for
21 which the "?" operators are employed, multiple indexes are
22 enumerated, and since $O(\log n)$ bits are required to
23 represent one index, $O(n \log n)$ bits are required to
24 represent all the indexes.

26 The "*" operator is the one that is attached to the child
27 element of a specific element in an element type
28 declaration when the child element does not appear or

1 appears more than once. And the "+" operator is the one
2 that is attached to the child element of a specific
3 element in an element type declaration when the child
4 element appears one or more times. Therefore, according
5 to XComp, when the "*" operator (or the "+" operator)
6 appears, the "*" (or the "+") operator is represented by
7 two choices one of which is maintained in the same state
8 or the other of which is shifted to different state, and
9 an index is provided for each choice. Upon the execution
10 of XComp, the index provided for a selected index is
11 output, and when the multiple elements for which "*"
12 operator is employed are present, multiple like indexes
13 are enumerated. Thus, since the number of indexes is
14 proportional to the number of elements that are present,
15 when n elements are present, $O(n)$ bits are required to
16 represent all the indexes.

17 That is, with XComp, the number of bits of code is
18 increased to represent a specific portion of the
19 structural portions that are not uniquely determined by
20 DTD, and a satisfactory compression efficiency can not
21 always be obtained.

22 SUMMARY OF THE INVENTION

23 The present invention overcomes the problems described
24 above. It is, therefore, one aspect of the invention to
25 provide a method for encoding XML data (XML documents)
26 that ensures a higher compression efficiency, and a method

1 and a system therefor. The decoding of encoded XML data
2 is performed when the order of the encoding processing is
3 reversed.

4 Furthermore, the encoding or decoding method of the
5 invention can be implemented as a system, and can also be
6 provided as a program that permits a computer to perform
7 the functions provided by the method.

8 BRIEF DESCRIPTION OF THE DRAWINGS:

9 These and other aspects, features, and advantages of the
10 present invention will become apparent upon further
11 consideration of the following detailed description of the
12 invention when read in conjunction with the following
13 drawing figures:

14 Fig. 1 is a functional block diagram showing an example
15 encoding system according to a first embodiment of the
16 present invention.

17 Fig. 2 is a functional block diagram showing an example
18 decoding system according to the first embodiment of the
19 present invention.

20 Fig. 3 is a functional block diagram showing an example
21 pre-process system according to a second embodiment of the
22 present invention.

1 Fig. 4 is a functional block diagram showing an example
2 post-process system according to the second embodiment of
3 the present invention.

4 Fig. 5 is a diagram showing a comparison of the encoding
5 efficiency provided by the encoding methods of the
6 embodiments and of XComp.

7 DESCRIPTION OF THE SYMBOLS

8 2-1: Syntax (type) generator
9 2-2: Separation unit
10 2-3: Syntax (value) generator
11 2-4: Transfer syntax generator
12 2-5: Compression unit
13 2-6: Combining unit
14 3-1: Syntax (type) generator
15 3-2: Transfer syntax decoder
16 3-3: Abstract syntax decoder
17 3-4: Combining unit
18 3-5: Separation unit
19 3-6: Decompression unit
20 9-1: DTD converter
21 9-2: XML data converter
22 9-3: Encoder
23 9-4: DTD converter
24 9-5: Decoder
25 9-6: XML data converter

1 DESCRIPTION OF THE INVENTION

2 The present invention overcomes the problems described
3 above. Even though the compression of XML documents,
4 XComp is the superior method. However, according to a
5 study performed by the present inventor, when XComp is
6 used to process XML data having a specific structure,
7 compression efficiency is reduced. That is, when the "?"
8 operator or the "*" operator (includes the "+" operator)
9 are employed for elements, compression efficiency is
10 reduced.

11 The "?" operator is the one that is attached to the child
12 element of a specific element in an element type
13 declaration when the child element does not appear or
14 appears only one time. According to XComp, when the "?"
15 operator appears the "?" operator is represented by
16 multiple choices each of which is shifted from one
17 specific state to another, and an index is provided for
18 the choice. Thereafter, when XComp is executed, an index
19 provided for a selected choice is output, and since the
20 number of available choices varies in consonance with how
21 many "?" operators follow an initial "?" operator, when n
22 "?" operators persist, for example, n+1 choices are
23 present for the first "?" operator. Therefore, n+1
24 indexes are required, and $O(\log n)$ bits are needed to
25 represent one index. Thus, when all the elements for
26 which the "?" operators are employed, multiple indexes are
27 enumerated, and since $O(\log n)$ bits are required to

1 represent one index, $O(n \log n)$ bits are required to
2 represent all the indexes.

3 The "*" operator is the one that is attached to the child
4 element of a specific element in an element type
5 declaration when the child element does not appear or
6 appears more than once. And the "+" operator is the one
7 that is attached to the child element of a specific
8 element in an element type declaration when the child
9 element appears one or more times. Therefore, according
10 to XComp, when the "*" operator (or the "+" operator)
11 appears, the "*" (or the "+") operator is represented by
12 two choices one of which is maintained in the same state
13 or the other of which is shifted to different state, and
14 an index is provided for each choice. Upon the execution
15 of XComp, the index provided for a selected index is
16 output, and when the multiple elements for which "*"
17 operator is employed are present, multiple like indexes
18 are enumerated. Thus, since the number of indexes is
19 proportional to the number of elements that are present,
20 when n elements are present, $O(n)$ bits are required to
21 represent all the indexes.

22 That is, with XComp, the number of bits of code is
23 increased to represent a specific portion of the
24 structural portions that are not uniquely determined by
25 DTD, and a satisfactory compression efficiency can not
26 always be obtained.

27 Thus the invention to provides methods, apparatus and

1 systems for encoding and decoding XML data (XML documents)
2 that ensures a higher compression and decompression
3 efficiency. Example embodiments of the present invention
4 are described in detail while referring to the
5 accompanying drawings. It should be noted, however, that
6 the present invention can be employed for various other
7 embodiments, and should not be limited to the embodiments
8 described herein. Throughout the embodiments, the same
9 reference numerals are used to denote corresponding or
10 identical components.

11 In the following embodiments, mainly a method or a system
12 will be described; however, as will be apparent to one
13 having ordinary skill in the art, the present invention
14 can be implemented as a computer-readable program.
15 Therefore, the present invention can be provided by
16 hardware or software, or a combination of hardware and
17 software, and can be stored on an arbitrary
18 computer-readable storage medium, such as a hard disk, a
19 CD-ROM, an optical storage device or a magnetic storage
20 device.

21 Furthermore, for the embodiments, an ordinary computer
22 system can be used that comprises commonly employed system
23 hardware resources, such as a central processing unit
24 (CPU), a main memory (RAM), a nonvolatile memory (ROM), a
25 co-processor, an image accelerator, a cache memory and an
26 input/output controller (I/O). An external storage
27 device, such as a hard disk drive, and communication
28 means, for connecting to a network such as the Internet,

1 can also be provided. Further, a variety of computer
2 types, including personal computers, workstations and main
3 frames, can be employed as the computer system.

4 First Example Embodiment

5 1. Premise conditions

6 Before beginning an explanation of the first embodiment,
7 premised conditions will now be described concerning the
8 processing performed during the embodiment.

9 (1) The structure of XML data is defined by a
10 grammar, which is designated for common use by a data
11 encoding side and a decoding side. A grammar
12 definition file, for example, is stored as external
13 data at a predetermined IP address, and the IP
14 address is used to refer to it when XML data is to be
15 encoded or decoded. In this embodiment, DTD is used
16 as the grammar definition, but this selection imposes
17 no limitation on which grammar can be used, and XML
18 Schema and RELAX, for example, can also be employed.

19 (2) XML data consists only of elements and text,
20 another method is used to manage others (e.g.,
21 attributes and process instructions). For example,
22 attributes and process instructions can be
23 represented as special elements and can be embedded
24 in the XML data, but in such a case, the grammar
25 should be altered. Further, attributes and process

1 instructions may be separated from the XML data, and
2 an XPointer used for their independent storage,
3 separate from elements and text.

4 (3) The text is compressed using a method such as
5 XMill.

6 Under these premised conditions, the encoding and
7 decoding of XML data for the embodiment will now be
8 described.

9 2. Encoding processing

10 2.1 Overview of the system configuration and the encoding 11 processing

12 Fig. 1 is a functional block diagram showing an example
13 encoding system according to the first embodiment of the
14 invention. The encoding system of this embodiment
15 comprises: a syntax (type) generator 2-1, a separation
16 unit 2-2, a syntax (value) generator 2-3, a transfer
17 syntax generator 2-4, a compression unit 2-5, and a
18 combining unit 2-6.

19 The syntax (type) generator 2-1 generates an ASN.1
20 abstract syntax (type) according to the DTD, and the
21 separation unit 2-2 separates the XML data into the
22 contents (text) of the element and the structure (the
23 element name and the structure). The syntax (value)
24 generator 2-3 generates an ASN.1 abstract syntax (value)
25 by employing the structure used for the elements, and the

1 transfer syntax generator 2-4 generates an ASN.1 transfer
2 syntax. The compression unit 2-5 compresses the separated
3 text, and the combining unit 2-6 combines the compressed
4 text and the ASN.1 transfer syntax to obtain encoded XML
5 data.

6 An explanation will now be given for an overview of the
7 encoding processing performed by the thus arranged
8 encoding system of the embodiment.

9 (Step 2-1) The syntax (type) generator 2-1 converts the
10 DTD into an ASN.1 abstract syntax (type).

11 (Step 2-2) The separation unit 2-2 separates the text
12 from the XML data that conforms to the DTD at step 2-1.
13 It should be noted that step 2-2 may be performed in
14 parallel with or before step 2-1.

15 (Step 2-3) The syntax (value) generator 203 converts the
16 XML data (the syntax of the elements), which is obtained
17 at step 2-2 by separating the text from the XML data, into
18 an ASN.1 abstract syntax (value) that conforms to the
19 ASN.1 abstract syntax (type) at step 2-1.

20 (Step 2-4) The transfer syntax generator 2-4 converts the
21 ASN.1 abstract syntax (value) obtained at step 2-3 into an
22 ASN.1 transfer syntax.

23 (Step 2-5) The compression unit 2-5 compresses the text
24 separated from the XML data at step 2-2. It should be

1 noted that step 2-5 is performed in parallel with steps
2 2-3 and 2-4.

3 (Step 2-6) The combining unit 2-6 combines the ASN.1
4 transfer syntax generated at step 2-4 and the text
5 compressed at step 2-5 to generate the encoded XML data.

6 It should be noted that the encoding rule determined by
7 the ASN.1 is employed to convert the ASN.1 abstract syntax
8 (value) into the ASN.1 transfer syntax. The encoding rule
9 can be, for example, BER, DER or PER. And since,
10 according to PER, the type or the value uniquely
11 determined according to the ASN.1 abstract syntax (type)
12 is not encoded, the encoding efficiency achieved is
13 especially high.

14 The above individual steps will now be explained in
15 detail.

16 2.2 Step 2-1

17 At step 2-1, the DTD is converted into an ASN.1 abstract
18 syntax (type). This method will now be described for each
19 pattern of the content model.

20 2.2.1 Element Content

21 The element content of the XML data is formed by the
22 combination of element names and operators. Generally,
23 the element names are represented by character strings.

1 The operators are the ones used for designating the order
2 in which the child elements in the element contents
3 appear, and the appearance frequencies of the elements.
4 As is well known, the ",", "|", "?", "*" and "+" operators
5 are permitted for DTD.

6 In principle, an element name is represented by an
7 identifier in the ASN.1 abstract syntax and an operator is
8 also represented as a type in the ASN.1 abstract syntax.
9 It should be noted, however, that since an identifier is
10 not encoded in accordance with BER an element name need
11 not be an identifier. An explanation will now be given
12 for the type used to represent each operator.

13 2.2.1.1 ",", " operator

14 In the ASN.1 abstract syntax, the defined type for the ",",
15 operator is "sequence". For example, when

```
16      <!ELEMENT a (b,c)>
17      <!ELEMENT b (#PCDATA)>
18      <!ELEMENT c (#PCDATA)>
```

19 is provided as the DTD, the ASN.1 abstract syntax (type)
20 is

```
21      A ::= SEQUENCE {
22          b B,
23          c C }
24      B ::= NULL
```


1 C ::= NULL,

2 where A, B and C are type references that are introduced
3 for convenience sake. Any type references may be employed
4 so long as no conflict arises. Further, since at step 2-2
5 the separation of the text occurs, the defined type for
6 both B and C is "null".

7 2.2.1.2 "|" operator

8 In the ASN.1 abstract syntax, the defined type for the "|"
9 operator is "choice". For example, when

10 <!ELEMENT a (b|c)>
11 <!ELEMENT b (#PCDATA)>
12 <!ELEMENT c (#PCDATA)>

13 is provided as the DTD, the ASN.1 abstract syntax (type)
14 is

15 A ::= CHOICE {
16 b B,
17 c C }
18 B ::= NULL
19 C ::= NULL.

20 2.2.1.3 "?" operator

21 In the ASN.1 abstract syntax, the assigned type for the
22 "?" operator is "sequence" and the keyword is "OPTIONAL".

1 For example, when

2 <!ELEMENT a (b?)>

3 <!ELEMENT b (#PCDATA)>

4 is provided as the DTD, the ASN.1 abstract syntax (type)
5 is

6 A ::= SEQUENCE {

7 b B OPTIONAL }

8 B ::= NULL.

9 2.2.1.4 "*" operator

10 In the ASN.1 abstract syntax, assigned type for the "*" 2025-10-28 10:40:00
11 operator is "sequence-of". For example, when

12 <!ELEMENT a (b*)>

13 <!ELEMENT b (#PCDATA)>

14 is provided as the DTD, the ASN.1 abstract syntax (type)
15 is

16 A ::= SEQUENCE OF B

17 B ::= NULL.

18 2.2.1.5 "+" operator

19 In the ASN.1 abstract syntax, the assigned type for the
20 "+" operator is "sequence-of" for a limited size. It

1 should be noted, however, that since the size limitation
2 has no effect on the encoding, the code for the "+"
3 operator is the same as that for the "*" operator. For
4 example, when

5 <!ELEMENT a (b+)>

6 <!ELEMENT b (#PCDATA)>

7 is provided as the DTD, the ASN.1 abstract syntax (type)
8 is

9 A ::= SEQUENCE SIZE (1 . . MAX) OF B

10 B ::= NULL.

11 2.2.1.6 No operator

12 In some cases, in the contents of an element only one
13 element name for which no operator is employed may be
14 designated. In the ASN.1 abstract syntax, the assigned
15 type in this case is "defined". For example, when

16 <!ELEMENT a (b)>

17 <!ELEMENT b (#PCDATA)>

18 is provided as the DTD, the ASN.1 abstract syntax (type)
19 is

20 A ::= B

21 B ::= NULL.

22 2.2.2. Mixed content

1 Mixed content is that which is obtained using the "|" operator to couple keyword "#PCDATA" with at least one element name, and by thereafter applying the "*" operator. In the ASN.1 abstract syntax, the types assigned to mixed content are "choice" and "sequence-of". For example, when

```
6      <!ELEMENT a (#PCDATA|b)*>
7      <!ELEMENT b (#PCDATA)>
```

8 is provided as the DTD, the ASN.1 abstract syntax (type) is

```
10     A ::= SEQUENCE OF CHOICE {
11         txt  NULL,
12         b    B }
13     B ::= NULL,
```

14 where txt is an identifier introduced for convenience sake that corresponds to the text included in the mixed content.

17 2.2.3 EMPTY

18 In the ASN.1 abstract syntax, the type assigned to an EMPTY is "null". For example, when

```
20      <!ELEMENT a EMPTY>
```

21 is provided as the DTD, the ASN.1 abstract syntax (type) is

1 A ::= NULL.

2 2.2.4 ANY

3 An ANY is equivalent to the mixed content that is
4 constituted by the keyword "#PCDATA" and all the element
5 names declared by the DTD. Therefore, the expression of
6 the ANY in the ASN.1 abstract syntax produces a mixed
7 content expression.

8 2.3 Step 2-2

9 At step 2-2, the text is separated from the XML data that
10 conforms to the DTD at step 2-1. For example, when the
11 XML data is

12 <a>
13 10
14 <c>20</c>
15 ,

16 "10" is separated from element b, and "20" is separated
17 from element c. As a result, the following XML data (the
18 element name and the structure) are obtained.

19 <a>
20
21 <c/>
22

1 The separated text can be collected for each element, and
2 can then be compressed like XMill, for example. Of
3 course, an alternate compression method can be employed.

4 2.4 Step 2-3

5 At step 2-3, the XML data at step 2-2 is converted into an
6 ASN.1 abstract syntax (value) that conforms to the ASN.1
7 abstract syntax (type) at step 2-1. For example, when the
8 ASN.1 abstract syntax (type) is

9 A:: = SEQUENCE {
10 b B,
11 c C }
12 B:: = NULL
13 C:: = NULL,

14 and when the following XML data (the element name and the
15 structure)

16 <a>
17
18 <c/>
19

20 are converted into an ASN.1 abstract syntax (value),

21 aA:: = {
22 b NULL,

1 c NULL }

2 is obtained.

3 2.5 Step 2-4

4 At step 2-4, the ASN.1 abstract syntax (value) at step 2-3
5 is converted into an ASN.1 transfer syntax in accordance
6 with the encoding rule determined by ASN.1. The encoding
7 rule is, for example, BER, DER or PER (ALIGNED/UNALIGNED),
8 of which PER (UNALIGNED) is preferable when increased
9 encoding efficiency is desired. However, BER, DER or PER
10 (ALIGNED) may also be employed.

11 2.6 Step 2-5

12 At step 2-5, the text is compressed. The well known LZ77
13 method and the well known technique that is described in a
14 subdivision of the background art are specifically
15 employed for the compression. The data compression may be
16 performed separately for individual elements, or
17 collectively for all the elements.

18 2.7 Step 2-6

19 At step 2-6, the ASN.1 transfer syntax is combined with
20 the compressed text. These two can be simply coupled, or
21 either a separator may be inserted between them, while
22 taking into consideration the separation process that will
23 be performed during decoding, or a header may be used to

1 which data bit count information has been added.

2 3. Decoding processing

3 3.1 Overview of the system configuration and the decoding 4 processing

5 Fig. 2 is a functional block diagram illustrating an
6 example decoding system according to the embodiment. The
7 decoding system of the embodiment comprises: a syntax
8 (type) generator 3-1, a transfer syntax decoder 3-2, an
9 abstract syntax decoder 3-3, a combining unit 3-4, a
10 separation unit 3-5 and a decompression unit 3-6.

11 The syntax (type) generator 3-1, as well as the syntax
12 (type) generator 2-1, converts the DTD into an ASN.1
13 abstract syntax (type). The separation unit 3-5 separates
14 encoded XML data to obtain an ASN.1 transfer syntax and
15 compressed text, and the transfer syntax decoder 3-2
16 converts the ASN.1 transfer syntax into an ASN.1 abstract
17 syntax (value) that conforms to the ASN.1 abstract syntax
18 (type). The abstract syntax decoder 3-3 then converts the
19 ASN.1 abstract syntax (value) into XML data (the element
20 name and the structure) that conforms to the DTD. The
21 combining unit 3-4 combines the decoded text (the contents
22 of the element) and the XML data (the element name and the
23 structure) and generates XML data, and the decompression
24 unit 3-6 decompresses the compressed text.

25 The order in which the decoding processing is performed,

1 while using the thus arranged decoding system of this
2 embodiment, is substantially the inverse of the order in
3 which the encoding processing was performed. An overview
4 of the decoding processing is as follows.

5 (Step 3-1) The DTD is converted into an ASN.1 abstract
6 syntax (type).

7 (Step 3-2) The encoded XML data is separated to obtain
8 the compressed text and the ASN.1 transfer syntax. It
9 should be noted that step 3-2 may be performed in parallel
10 with or preceding step 3-1.

11 (Step 3-3) The ASN.1 transfer syntax is converted into an
12 ASN.1 abstract syntax (value) that conforms to the ASN.1
13 abstract syntax (type) at step 3-1.

14 (Step 3-4) The ASN.1 abstract syntax (value) obtained at
15 step 3-3 is converted into XML data (the element name and
16 the structure) that conforms to the DTD at step 3-1.

17 (Step 3-5) The compressed text obtained at step 3-2 is
18 decompressed.

19 (Step 3-6) The XML data (the element name and the
20 structure) obtained at step 3-4 is combined with the text
21 decompressed at step 3-5.

22 Since at steps 3-1 to 3-6 the decoding processes that are
23 performed are obvious because they are inversions of the

1 corresponding encoding processes, a detailed explanation
2 will not be given for these processes.
3 The following is a specific example of the conversion of
4 DTD and of the use of PER to encode XML data that conform
5 to the DTD.

6 4. Case wherein the "," operator is included

7 In this case, the DTD that includes the "," operator and
8 the XML data that conform to the DTD are employed. When

9 <!ELEMENT a (b,c)>
10 <!ELEMENT b (#PCDATA)>
11 <!ELEMENT c (#PCDATA)>

12 is provided as the DTD, at step 2-1, the ASN.1 abstract
13 syntax (type) that is generated is

14 A ::= SEQUENCE {
15 b B,
16 c C }
17 B ::= NULL
18 C ::= NULL.

19 When the XML data to be encoded that conform to this DTD
20 are

21 <a>
22 10
23 <c>20</c>

1 ,

2 at step 2-2 the element (the element name and the
3 structure of the XML data) obtained is

4 <a>

5

6 <c/>

7 ,

8 and the ASN.1 abstract syntax (value), which at step 2-3
9 is generated from this element, is

10 aA ::= {
11 b NULL,
12 c NULL} .

13 According to PER, in principle, the values of elements of
14 the "sequence" type are encoded in order. However, since
15 the values of b and c are "null", and "null" is encoded as
16 an empty bit string, the value acquired by a is an empty
17 bit string. In this case, the obtained code is obtained
18 exceptionally as

19 00000000₍₂₎ .

20 The ASN.1 transfer syntax is generated in this manner. It
21 should be noted that the generation of the ASN.1 transfer
22 syntax is performed at step 2-4, and that the subscript
23 (2) indicates that the code represents a binary number.

1 Thereafter, the text compressed at step 2-5 and the ASN.1
2 transfer syntax (00000000₍₂₎) are combined at step 2-6, and
3 encoded XML data is generated.

4 5. Case wherein the "|" operator is included

5 In this case, the DTD that includes the "|" operator and
6 the XML data that conforms to the DTD are employed. When

7 <!ELEMENT a (b|c)>
8 <!ELEMENT b (#PCDATA)>
9 <!ELEMENT c (#PCDATA)>

10 is provided as the DTD, at step 2-1 the ASN.1 abstract
11 syntax (type) generated is

12 A ::= CHOICE {
13 b B,
14 c C }
15 B ::= NULL
16 C ::= NULL.

17 When the XML data to be encoded that conforms to this DTD
18 is

19 <a>
20 10
21 ,

22 at step 2-2 the element (the element name and the
23 structure of the XML data) that is obtained is

1 <a>
2
3 .

4 The ASN.1 abstract syntax (value), which at step 2-3 is
5 generated from this element, is

6 aA ::= b: NULL.

7 According to PER, as the value of the "choice" type, the
8 index (0 base) of a selected element is encoded, and then
9 the value of the element is encoded. In this case, since
10 b is selected, the index is 0, and the value of b is null.
11 Thus, the value obtained for a is

12 0XXXXXXX₍₂₎,

13 where each X denotes a padding bit that is added in order
14 to obtain a multiple of 8 bits. In this manner, the ASN.1
15 transfer syntax is generated. It should be noted that the
16 generation of compressed text and the combining of the
17 compressed text and the ASN.1 transfer syntax are
18 performed in the same manner as in "4. Case wherein the
19 "," operator is included".

20 6. Case wherein the "?" operator is included

21 In this case, the DTD that includes the "?" operator and
22 the XML data that conforms to the DTD are employed. When

```

1      <!ELEMENT a (b?,c)>
2      <!ELEMENT b (#PCDATA)>
3      <!ELEMENT c (#PCDATA)>

```

4 is provided as the DTD, the ASN.1 abstract syntax (type)
5 that is generated at step 2-1 is

```

6      A ::= SEQUENCE {
7          id0 SEQUENCE {
8              b B OPTIONAL },
9              c C }
10     B ::= NULL
11     C ::= NULL.

```

12 When the XML data to be encoded that conforms to this DTD
13 is

```

14     <a>
15         <b>10</b>
16         <c>20</c>
17     </a>,

```

18 the element (the element name and the structure of the XML
19 data) that is obtained at step 2-2 is

```

20     <a>
21         <b/>
22         <c/>
23     </a>.

```

1 The ASN.1 abstract syntax (value), which at step 2-3 is
2 generated from this element, is

```
3      aA ::= {  
4          id0 {  
5              b  NULL }  
6              c  NULL }.
```

7 According to PER, when the designation "OPTIONAL" is
8 applied for at least one of the "sequence" type elements,
9 a bit string used to represent the elements that are
10 present is added before the values of the elements are
11 encoded. When an element is present, a bit in the bit
12 string is set to 1, while if the element is not present,
13 the bit is reset to 0. In this case, since the
14 designation "OPTIONAL" is applied only for b, and since
15 this element is present, first, a bit string containing a
16 bit set to 1 is added. Then, the values of b and c are
17 encoded, and since for both a "null" result is obtained,
18 the value assigned to a is
19 1XXXXXXX₍₂₎.

20 In this manner, the ASN.1 transfer syntax is generated.
21 It should be noted that the generation of compressed text
22 and the combining of the compressed text and the ASN.1
23 transfer syntax are performed in the same manner as in "4.
24 Case wherein the "," operator is included".

25 7. Case wherein the "*" operator is included

1 In this case, the DTD that includes the "*" operator and
2 the XML data that conforms to the DTD are employed. When
3 <!ELEMENT a (b*)>
4 <!ELEMENT b (#PCDATA)>
5 is provided as the DTD, the ASN.1 abstract syntax (type)
6 generated at step 2-1 is

7 A ::= SEQUENCE OF B
8 B ::= NULL.

9 When the XML data to be encoded that conform to this DTD
10 are

11 <a>
12 10
13 20
14 ,

15 the element (the element name and the structure of the XML
16 data) obtained at step 2-2 is

17 <a>
18
19
20 .

21 The ASN.1 abstract syntax (value), which at step 2-3 is
22 generated from this element, is

1 aA::= {
2 NULL,
3 NULL}.

4 In accordance with PER, to obtain the "sequence-of" type
5 value, first, the number of elements is encoded and then
6 the values of these elements are encoded in order. In
7 this case, since the number of elements is two, a value of
8 2 is encoded. Then, since the values of the elements are
9 encoded and the result obtained in each case is "null",
10 the value of a is 00000010₍₂₎.

11 In this manner, the ASN.1 transfer syntax is generated.
12 It should be noted that the generation of compressed text
13 and the combining of the compressed text and the ASN.1
14 transfer syntax are performed in the same manner as in "4.
15 Case wherein the "," operator is included".

16 8. Case wherein mixed content is included

17 In this case, the DTD that includes mixed content and the
18 XML data that conforms to the DTD are employed. When

19 <!ELEMENT a (#PCDATA|b)*>
20 <!ELEMENT b (#PCDATA)>

21 is provided as the DTD, at step 2-1 the ASN.1 abstract
22 syntax (type) that is generated is

23 A::= SEQUENCE OF CHOICE {

1 txt NULL,
2 b NULL}.

3 When the XML data to be encoded that conform to this DTD
4 are

5 <a>
6 xxx
7 10
8 20
9 ,

10 the element (the element name and the structure of the XML
11 data) obtained at step 2-2 is

12 <a>
13 <txt/>
14
15
16 .

17 It should be noted that "xxx" represents the contents of
18 the text, and "txt" represents an element that is the
19 text. The ASN.1 abstract syntax (value), which is
20 generated from this element at step 2-3, is

21 aA ::= {
22 txt: NULL,
23 b: NULL,
24 c: NULL}.

1 The values of the "sequence-of" type and of the "choice"
2 type are encoded in the above described manner. In this
3 example, the number of elements is three, and since txt, b
4 and b, in the named order, are selected as elements, the
5 indexes are 0, 1 and 1. Since all these values are null,
6 the value of a is

7 00000011₍₂₎
8 011XXXXX₍₂₎.

9 In this manner, the ASN.1 transfer syntax is generated.
10 It should be noted that the generation of compressed text
11 and the combining of the compressed text and the ASN.1
12 transfer syntax are performed in the same manner as in "4.
13 Case wherein the "," operator is included".

14 Second Example Embodiment

15 In the first embodiment, an explanation was given in which
16 the assumption was that the XML data was constituted
17 merely by elements and text, and another method is used to
18 manage others (e.g., an attribute and a process
19 instruction). In the second embodiment, example
20 processing for XML data that includes an attribute and a
21 process instruction will be described. That is, a method
22 will be explained whereby special elements are used to
23 represent an attribute and a process instruction using
24 special elements, and to embed them in the XML data. In
25 this case, it is required the grammar is changed. It

1 should be noted that, regardless of the example given in
2 this embodiment, a method may also be employed whereby an
3 attribute and a process instruction are separated from XML
4 data and stored using an XPointer. The method used for
5 embedding an attribute and a process instruction as
6 special elements in the XML data can be understood as
7 being a pre-process and a post-process for the processing
8 of the first embodiment. As a result, XML data having a
9 greater range can be handled. As an example pre-process,
10 an explanation will now be given for a method employed for
11 embedding, in XML data, entries that are defined by the
12 DTD.

13 9.1 Overview of the pre-process system and the 14 pre-process

15 Fig. 3 is a functional block diagram showing an example
16 pre-process system according to the embodiment. The
17 pre-process system of this embodiment comprises a DTD
18 converter 9-1, an XML data converter 9-2 and an encoder
19 9-3. The DTD converter 9-1 converts a representative DTD
20 into a DTD' while taking entries other than the element
21 into account. The XML data converter 9-2 converts the XML
22 data that conforms to the representative DTD into XML
23 data' that conforms to the DTD'. The encoder 9-3 is the
24 encoding system employed for the first embodiment.

25 The pre-process method used for this embodiment comprises
26 the steps: conversion of the DTD by the DTD converter 9-1,
27 and conversion of the XML data by the XML data converter

1 9-2. Then, when following these conversions the encoding
2 process in the first embodiment is performed, encoded XML
3 data is generated.

4 9.2 DTD conversion process

5 The DTD includes not only an element type declaration but
6 also three other declarations: an attribute-list
7 declaration, an entity declaration, and a notation
8 declaration.

9 The entities defined in the entity declaration are a
10 parsed entity, an unparsed entity, and a parameter entity.
11 The parsed entity can be referred to, in text or an
12 attribute value, at any location, and is simply expanded.
13 Since the unparsed entity can only be referred to by an
14 attribute value, so long as the attribute can be
15 processed, the operation is satisfactory. But since the
16 parameter entity can be referred to only in the DTD, the
17 operation is not considered in this embodiment.

18 Since the notation defined in the notation declaration can
19 be referred to merely by using the attribute value, so
20 long as the attribute can be processed, the operation is
21 satisfactory. As a result, the performance of the
22 pre-process for the DTD is a precursor of the performance
23 of the pre-process for the element type declaration and of
24 the attribute-list declaration.

25 In the attribute-list declaration, an attribute to be

1 provided for a specific element and the value that the
2 attribute can obtain are defined. As needed, a default
3 value is also defined. The provision of the attribute for
4 an element varies, depending on whether the attribute is
5 REQUIRED or IMPLIED or on whether a default value is
6 defined and whether it is FIXED. Therefore, while, in
7 principle, the attribute is represented as an element, it
8 is preferable that the expression of the attribute be
9 changed more or less in accordance with the definition.

10 The rules for changing the element type declaration in
11 accordance with the attribute-list declaration are as
12 follows.

13 (1) An attribute is represented as an element
14 (hereinafter referred to as an attribute element),
15 and the element name is uniquely determined from the
16 attribute name, e.g., "the element name of a parent"
17 + "_" + "attribute name". It should be noted,
18 however, that the element name is not yet used.
19 Thereafter, this naming rule is employed to determine
20 the element name.

21 (2) All of the attribute values are handled as
22 "CDATA", and are included in the attribute element.

23 (3) The attribute element is inserted into the first
24 of the child elements of the parent element.

25 (4) The REQUIRED attribute is represented as an

1 element.

2 (5) The IMPLIED attribute is represented as an
3 element for which the "?" operator is employed.

4 (6) The attribute for which a default value is
5 defined is represented as an element for which the
6 "?" operator is employed.

7 (7) The attribute for which the FIXED default value
8 is defined is ignored.

9 A specific example will now be described. For example,
10 when

```
11 <!ELEMENT a (b,c)>
12 <!ATTLIST a w CDATA #REQUIRED
13                x ID # IMPLIED
14                y (0|1|2) "0"
15                z CDATA #FIXED "abc">
```

16 is provided as the DTD, the DTD' is

```
17 <!ELEMENT a (a_w,a_x?,a_y?,b,c)>
18 <!ELEMENT a_w (#PCDATA)>
19 <!ELEMENT a_x (#PCDATA)>
20 <!ELEMENT a_y (#PCDATA)>.
```

21 9.3 XML data conversion process

1 The XML data that conforms to the DTD is converted into
 2 XML data that conforms to the DTD'. Basically, the
 3 attribute provided for an element need only be converted
 4 into an attribute element, and the obtained attribute
 5 element inserted into the first of the child elements of
 6 the element. It should be noted that, when the default
 7 value is defined for the attribute and the attribute value
 8 matches the default value, the attribute is not converted
 9 into an attribute element.

10 When, for example,

```
11      <aw = "xyz"  y = "0"  z = "abc">
12          <b> . . . </b>
13          <c> . . . </c>
14      </a>
```

15 is employed as XML data that conforms to the DTD in 9.2,
 16 the XML data' is converted into

```
17      <a>
18          <a_w>xyz</a_w>
19          <b> . . . </b>
20          <c> . . . </c>
21      </a>.
```

22 Since the attribute x is not provided for the element a,
 23 the element a_x does not appear. Further, since the value
 24 of the attribute y matches the default value, the element
 25 a_y does not appear, either. In addition, since the FIXED

1 default value is defined for the attribute z, it is
2 deleted.

3 9.4 Overview of the post-process system and the 4 post-process

5 Fig. 4 is a functional block diagram showing an example
6 post-process system according to the embodiment. The
7 post-process system of this embodiment comprises: a DTD
8 converter 9-4, a decoder 9-5, and an XML data converter
9 9-6. The DTD converter 9-4 is the same as the DTD
10 converter 9-1, and the decoder 9-5 is the decoding system
11 used in the first embodiment. The XML data converter 9-6
12 converts the XML data' that conforms to the DTD' into XML
13 data that conforms to the DTD.

14 The post-process method of this embodiment comprises the
15 steps, performed following the completion of the decoding
16 process in the first embodiment, for the DTD conversion
17 performed by the DTD converter 9-4 and for the XML data
18 conversion performed by the XML data converter 9-6. Since
19 the DTD conversion performed by the DTD converter 9-4 is
20 the same as the one employed in 9.2, and since for the XML
21 data conversion the XML data converter 9-6 performs a
22 process that is the inverse of the one employed for the
23 conversion in 9.3, detailed explanations for these
24 operations will not be given. A broken-line arrow in Fig.
25 4 indicates that the DTD can be referred to as needed, and
26 so long as the attribute element can be identified and the
27 attribute name can be uniquely determined from the name of

1 the attribute element, the DTD need not be referred to.
2 Furthermore, since the post-process depends on how the
3 pre-process has been performed, it is natural for the
4 post-process of this embodiment to be changed when the
5 pre-process differs.

6 10. Effects obtained by the embodiments

7 By employing the encoding method of the first or the
8 second embodiment, the XML data can be efficiently
9 compressed. The effects obtained by the embodiments will
10 now be described while being compared with the effects
11 obtained by XComp.

12 The encoding method of this invention is the same as XComp
13 because, in XML data encoding, the information uniquely
14 determined by the DTD is not encoded. However, the
15 compression efficiency obtained with XComp is reduced when
16 XML data has a specific structure. Specifically, the
17 compression efficiency is reduced when the "?" operator or
18 the "*" operator (includes the "+" operator) is employed
19 for the element.

20 10.1 "?" operator

21 According to the method of this invention, a bit string is
22 employed to indicate the presence of an element, while
23 according to XComp, indexes provided for each existing
24 elements (choices) are enumerated. This difference is
25 particularly remarkable when several elements for which

1 the "?" operator is employed persist and all continue to
2 be present.

3 For example, assume XML data:

```
4      <a>
5          <b>xxx</b>
6          <c>yyy</c>
7      </a>
```

8 that conforms to DTD:

```
9      <!ELEMENT a (b?,c?)>
10     <!ELEMENT b (#PCDATA)>
11     <!ELEMENT c (#PCDATA)>.
```

12 In this XML data, elements b and c coexist. The XML data
13 is encoded into 11₍₂₎ using the method of the invention, and
14 to simplify the explanation, a padding bit is not added.

15 As for the XComp, when automaton:

16		S1	S2	S3	S4
17	S1	-	b/1	c/2	ϵ /3
18	S2	-	-	c/1	ϵ /2
19	S3	-	-	-	ϵ / ϵ
20	S4	-	-	-	-

21 is prepared, an index string of 11 is obtained. And when

1 this index string is converted into a 0 base and the 0
2 base is then encoded by using the minimum required number
3 of bits, code $000_{(2)}$ is obtained. Subsequently, when the
4 code obtained by the method of the invention is compared
5 with the code obtained by XComp, it is found that the code
6 obtained by employing this invention is shorter by one
7 bit.

8 Table 1 shows the results obtained with the method of the
9 invention compared with the results obtained with XComp,
10 when the objective is an evaluation of the number of bits
11 required for encoding several elements, in which the "?"
12 operator is employed, that persist, continue to be
13 present. As is apparent from table 1, the encoding
14 efficiency provided by the method of the invention is
15 greater than is that provided by XComp. Generally, when
16 there are n persistent elements for which the "?" operator
17 is employed, Xcomp requires $O(n \log n)$ bits, while the
18 method of the invention requires only $O(n)$ bits.

19 [Table. 1]

	Xcomp(prior art)			Present invention
The number of ?s	The number of choices	The number of bits	The total number of bits	The number of bits
1	2	1	1	1
2	3,2	2,1	3	2
3	4,3,2	2,2,1	5	3
4	5,4,3,2	3,2,2,1	8	4
5	6,5,4,3,2	3,3,2,2,1	11	5

1 10.2 "*" operator

2 According to the method of this invention, the number of
3 existing elements is encoded, while according to the
4 XComp, indexes provided for the existing elements
5 (choices) are listed. This difference is particularly
6 remarkable when many elements are present in which the "*"
7 operator is employed.

8 For example, assume XML data:

9 <a>
10 xxx
11 yyy
12 . . .
13 zzz
14

15 that conforms to DTD:

16 <!ELEMENT a (b*)>
17 <!ELEMENT b (#PCDATA)>.

18 In this case, further assume that the element b has
19 appeared eight times. Using the method of the invention,
20 the XML data is encoded into 00001000₍₂₎ by the method of
21 the invention.
22 As for the XComp, when automaton:

23 S0 S1

1 S0 b/1 ε/2
2 S1 - -

3 is prepared, an index string of 111111112 is obtained.
4 And when this index string is converted into 0 base and
5 the 0 base is then encoded by using the minimum required
6 number of bits, code 000000001₍₂₎ is obtained. Thereafter,
7 when the code obtained by the method of the invention is
8 compared with the code obtained by XComp, it is found that
9 the code obtained by this invention is shorter by one bit.

10 Fig. 5 is a graph showing a comparison between the method
11 of the invention and XComp of the number of bits required
12 to encode many elements that are present for which the "*"
13 operator is employed. As is apparent from Fig. 5, when
14 eight or more elements are present, these elements can be
15 encoded more efficiently when the method of the invention
16 is used than when XComp is used. The method of the
17 invention requires a number of bits having the same order
18 as in XComp; but in actuality, a smaller number of bits is
19 sufficient.

20 The present invention has been specifically explained by
21 using the described embodiments. However, the invention
22 is not limited to these embodiments and can be variously
23 modified without departing from the scope of the
24 invention. For example, the method of the invention may
25 be employed by combining multiple operators. There are
26 four operators ",", "|", "?" and "*" (includes "+") that
27 can be used for the content model. Therefore, while

1 taking the employment order into account, there are 16
2 operator combinations, and for each combination, XML data
3 can be converted into an ASN.1 abstract syntax in the same
4 manner as in the embodiments. Thus, XML data that
5 includes any combination of operators can be encoded when
6 the invention is used.

7 One of the points applicable to the method explained in
8 these embodiments is the definition of the grammar, and
9 the expression, in ASN.1 abstract syntax, of the XML data
10 that conforms to the grammar. However, another expression
11 may also be employed.

12 For example, in the above embodiments, the "?" operator is
13 represented by using a combination of the "sequence" type
14 and the keyword "OPTIONAL". Specifically, the DTD:

15 <!ELEMENT a (b?,c)>

16 is converted into an ASN.1 abstract syntax:

17 A ::= SEQUENCE {
18 id0 SEQUENCE {
19 b B OPTIONAL },
20 c C }.

21 The DTD may also be converted into an ASN.1 abstract
22 syntax:

23 A ::= SEQUENCE {

1 b B OPTIONAL,
2 c C }.

3 This conversion is convenient because the code obtained by
4 BER or DER is shorter. However, since this conversion
5 method does not function well for the following DTD:

6 <!ELEMENT a (b?)>
7 <!ELEMENT a (b?|c)> ,

8 the representation as in the embodiments is more
9 appropriate.

10 Further, in the embodiments, each of the operators is
11 represented by one of the types. For example, the ","
12 operator is represented by the "sequence" type, and the
13 "|" operator is represented by the "choice" type. In
14 addition, another method can be employed according to
15 which the use of an operator is regarded as "production",
16 and for employing the "sequence" type to represent all the
17 operators. According to this method, the "|" operator and
18 the "*" operator (includes the "+" operator), which are
19 not represented by the sequence type, and the case wherein
20 no operator is present can be represented as follows.

21 <!ELEMENT a(b|c)> can be represented as

22 A ::= SEQUENCE {
23 id0 CHOICE {
24 b B,
25 c C } };

1 <!ELEMENT a(b*)> can be represented as
2 A:: = SEQUENCE {
3 b SEQUENCE OF B }; and
4 <!ELEMENT a(b)> can be represented as
5 A::= SEQUENCE {
6 b B }.

7 Since according to the expression method all the operators
8 are represented by the "sequence" type, the operators can
9 be easily identified visually. Further, since the
10 representation method corresponds to the wrapping of the
11 operator by the "sequence" type, this method can function
12 well even with a combination of multiple operators.
13 However, since the types are always encoded by BER or DER,
14 the length of the code is increased each time a type is
15 inserted. Therefore, more types than necessary should not
16 be inserted, and for this reason, it is appropriate that
17 the operators be represented by employing the method of
18 the invention.

19 Advantageous effects obtained by this invention include
20 the following consideration. The XML data encoding
21 (compression) efficiency can be improved, and XML data,
22 including descriptions other than the elements such as
23 attributes, can be encoded (compressed). Thus, the
24 communication load imposed by the transfer of XML data can
25 be reduced, as can the capacity of the storage areas that
26 are used for XML data.
27 The present invention can be realized in hardware, software,
28 or a combination of hardware and software. A visualization

1 tool according to the present invention can be realized in a
2 centralized fashion in one computer system, or in a
3 distributed fashion where different elements are spread
4 across several interconnected computer systems. Any kind of
5 computer system - or other apparatus adapted for carrying
6 out the methods and/or functions described herein - is
7 suitable. A typical combination of hardware and software
8 could be a general purpose computer system with a computer
9 program that, when being loaded and executed, controls the
10 computer system such that it carries out the methods
11 described herein. The present invention can also be
12 embedded in a computer program product, which comprises all
13 the features enabling the implementation of the methods
14 described herein, and which - when loaded in a computer
15 system - is able to carry out these methods.

16 Computer program means or computer program in the present
17 context include any expression, in any language, code or
18 notation, of a set of instructions intended to cause a
19 system having an information processing capability to
20 perform a particular function either directly or after
21 conversion to another language, code or notation, and/or
22 after reproduction in a different material form.

23 Thus the invention includes an article of manufacture
24 which comprises a computer usable medium having computer
25 readable program code means embodied therein for causing a
26 function described above. The computer readable program
27 code means in the article of manufacture comprises
28 computer readable program code means for causing a

1 computer to effect the steps of a method of this
2 invention. Similarly, the present invention may be
3 implemented as a computer program product comprising a
4 computer usable medium having computer readable program
5 code means embodied therein for causing a a function
6 described above. The computer readable program code means
7 in the computer program product comprising computer
8 readable program code means for causing a computer to
9 effect one or more functions of this invention.
10 Furthermore, the present invention may be implemented as a
11 program storage device readable by machine, tangibly
12 embodying a program of instructions executable by the
13 machine to perform method steps for causing one or more
14 functions of this invention.

15 It is noted that the foregoing has outlined some of the
16 more pertinent objects and embodiments of the present
17 invention. This invention may be used for many
18 applications. Thus, although the description is made for
19 particular arrangements and methods, the intent and
20 concept of the invention is suitable and applicable to
21 other arrangements and applications. It will be clear to
22 those skilled in the art that modifications to the
23 disclosed embodiments can be effected without departing
24 from the spirit and scope of the invention. The described
25 embodiments ought to be construed to be merely illustrative
26 of some of the more prominent features and applications of
27 the invention. Other beneficial results can be realized by
28 applying the disclosed invention in a different manner or
29 modifying the invention in ways known to those familiar with

1 the art.

2025-10-20 09:56:40